Queues

Queue at grocery store (FIFO)

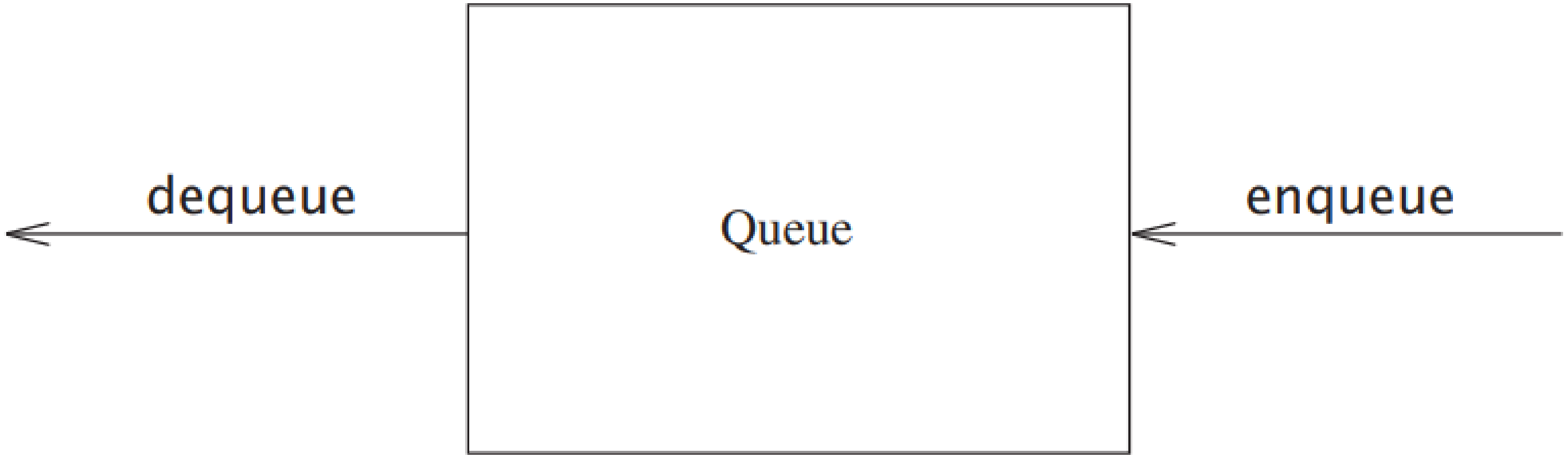
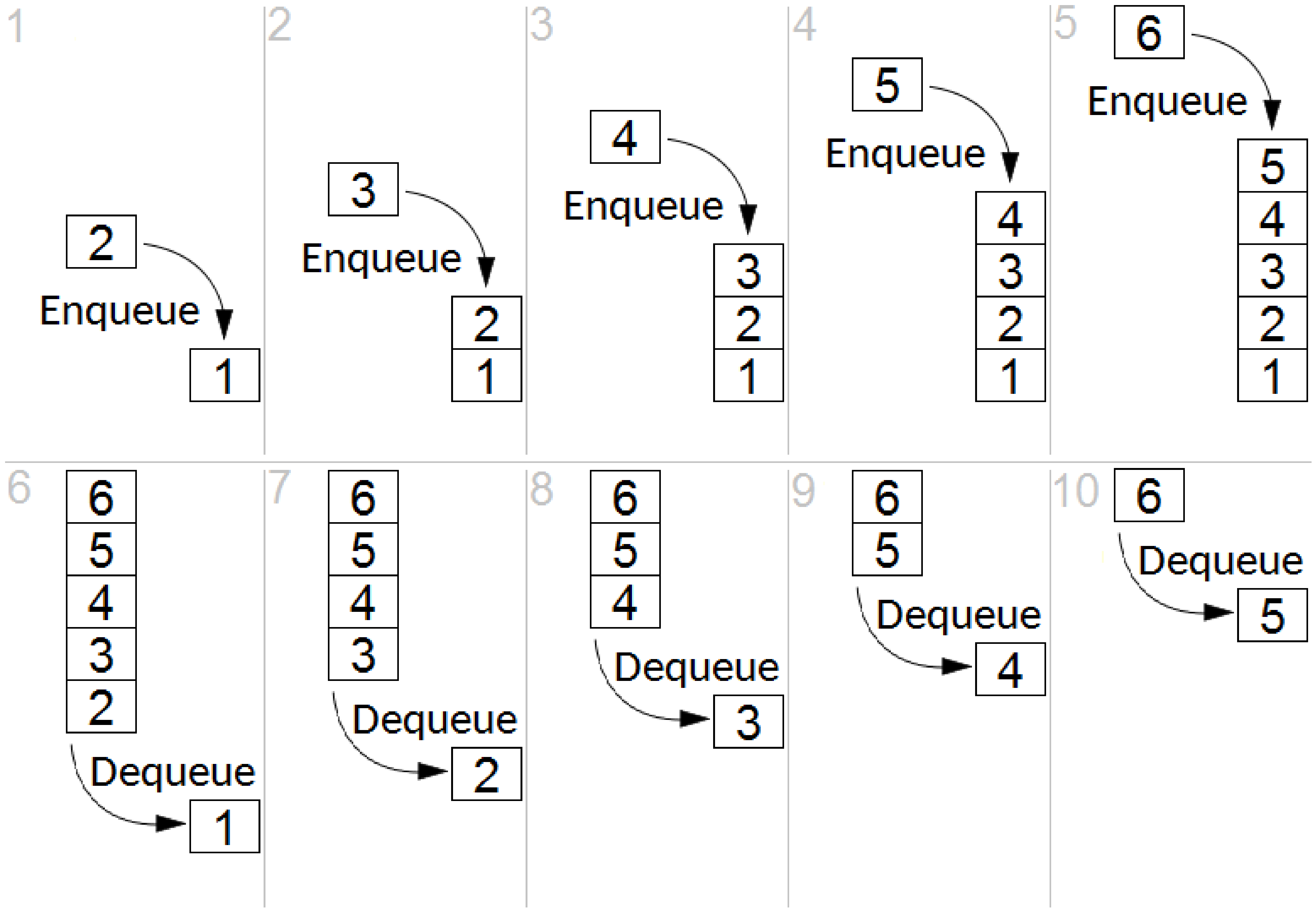


Figure 3.27 Model of a queue

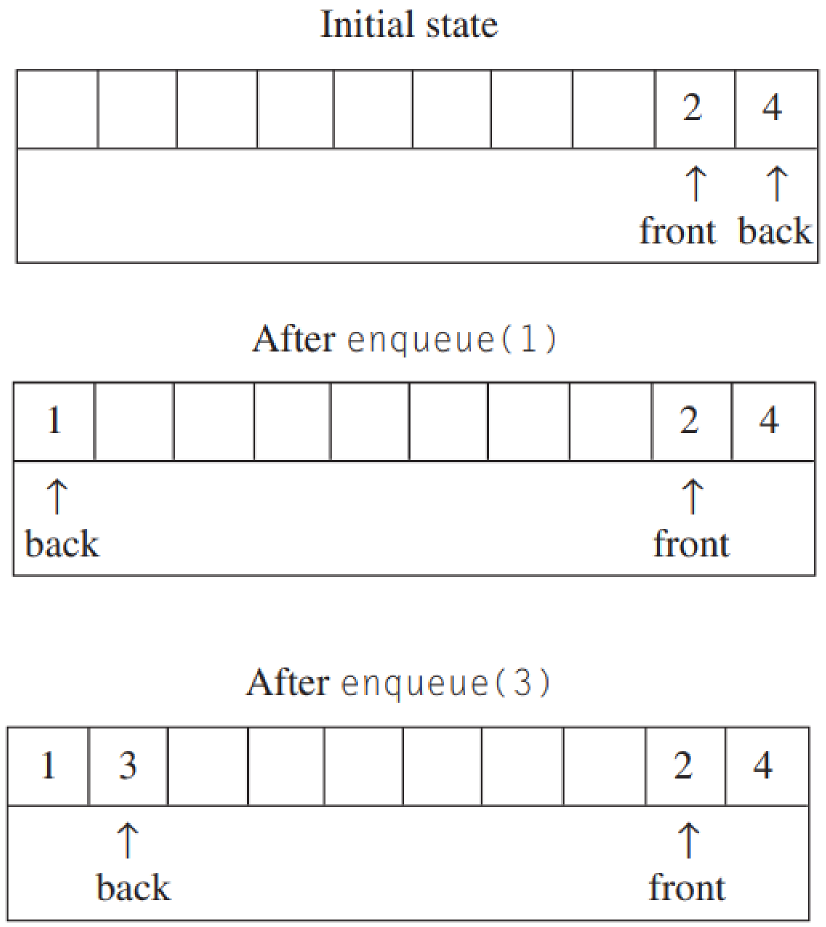

1
2
Enqueue
1
2
3
Enqueue
2
1
3
4
Enqueue
3
2
1
4
5
Enqueue
4
3
2
1
5
6
Enqueue
5
4
3
2
1
6
6
5
4
3
2
Dequeue
1
7
6
5
4
3
Dequeue
2
8
6
5
4
Dequeue
3
9
6
5
Dequeue
4
10
6
Dequeue
5

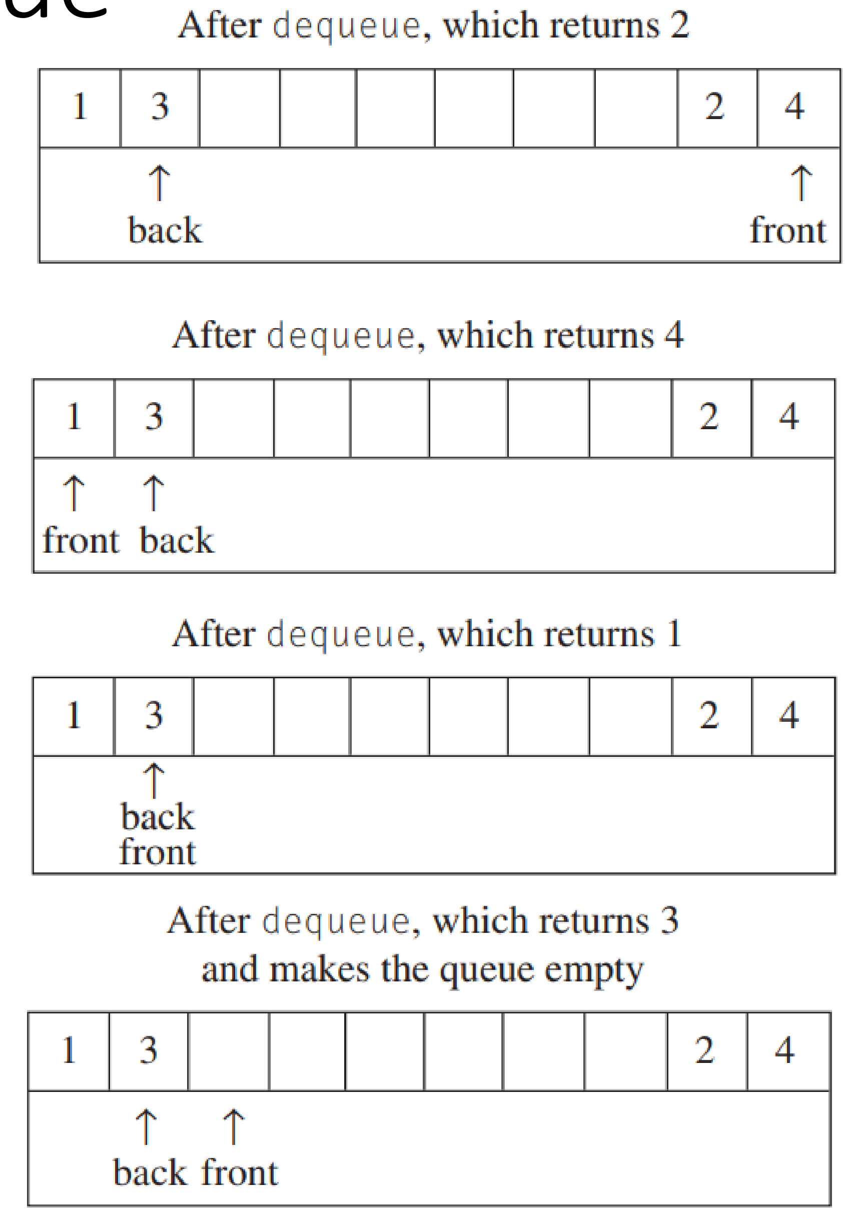
Implementing a Queue ADT

- As with stack, can use List ADT and restrict set of operations to only the subset needed for Queue
- Basic operations:
  - Enqueue (insert element at end of list)
  - Dequeue (remove and delete element from front of list)
- Both linked list and array implementation give $O(1)$ for enqueue and dequeue

Array implementation of Queue

- 4 variables:
- `array`
- `size`
- `front`
- `back`



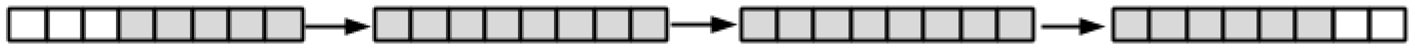

After dequeue, which returns 2
1
3
2
4
back
front
After dequeue, which returns 4
1
3
2
4
front
back
After dequeue, which returns 1
1
3
2
4
back
front
After dequeue, which returns 3
and makes the queue empty
1
3
2
4
back
front

Deque (double-ended queue)

- Deques are containers with dynamic size that can be expanded or contracted on both ends
- Allow for efficient random access using `operator[]` (like vectors)
- However, not guaranteed to store all elements in contiguous storage locations (cannot simply offset pointer to get next element)
  - Vectors use a single array that is occasionally reallocated for growth
  - Elements of deque can be scattered throughout memory
- Can grow more efficiently than vector under certain circumstances (when the sequence is long)

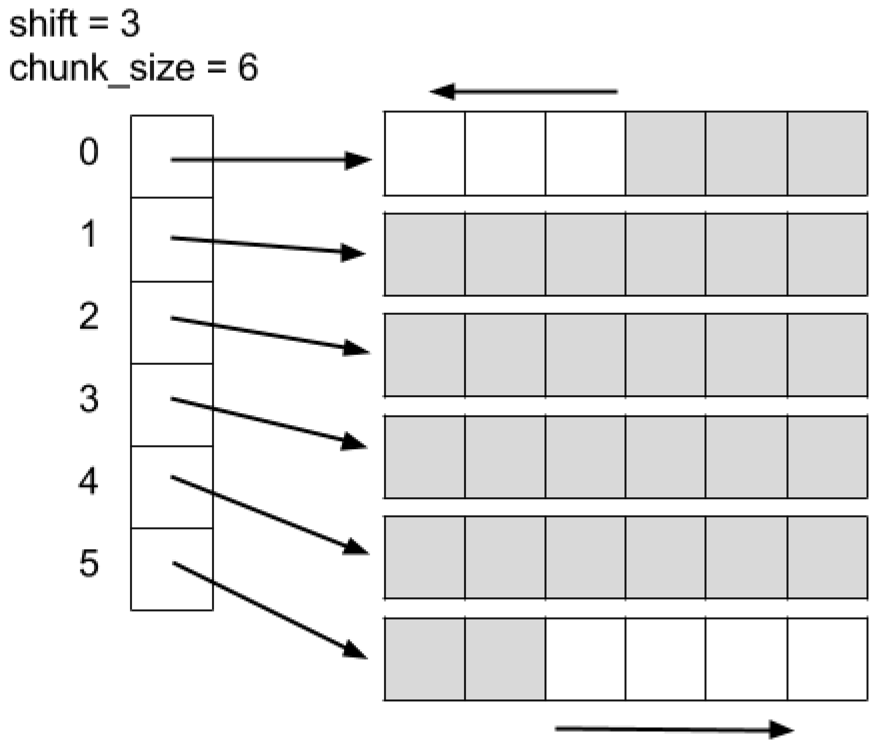
Stl::deque implementation

a)



- Stl::deque implemented as dynamic array composed of linked arrays (a)
  - Gives fast insert and remove on both ends (`push_front`, `pop_front`, `push_back`, `pop_back` all $O(1)$)
  - `operator[]` is also $O(1)$
- `push_back` adds element to last chunk, or allocates a new chunk if necessary (b)
  - Similar for `push_front`

b)



```
#include<vector>
#include<iostream>

template<typename T>
class ArrayQueue {

public:
ArrayQueue(int maxCapacity = 10) :
currentSize{ 0 }, front{ 0 }, back{ 0 }, arr(maxCapacity) {}

void enqueue(T elem)
{
if (currentSize < arr.capacity())
{
arr[back] = elem;
back = ++back % arr.capacity();
}
else
std::cout << "queue is already full\n";
}

T dequeue()
{
T elem = std::move(arr[front]);
front = ++front % arr.capacity();
return elem;
}

void printQueue()
{
std::cout << "[";
for (int i = 0; i < arr.capacity(); ++i)
{
if (i == front)
std::cout << "front:";
if (i == back)
std::cout << "back:";
std::cout << arr[i] << " ";

}
std::cout << "\n";
}



private:

std::vector<T> arr;
int currentSize;
int front;
int back;


};

int main()
{

ArrayQueue<int> q{ 5 };
q.enqueue(3);
q.enqueue(4);
q.enqueue(5);
q.enqueue(6);
q.enqueue(7);
std::cout << q.dequeue() << std::endl;
std::cout << q.dequeue() << std::endl;
q.enqueue(8);
q.printQueue();

}
```